Bloc Fi-Math3
Optimisation continue

S. ROBERT

Travaux Dirigés

Méthodes d'optimisation classiques

OBJECTIFS

- **Implémenter et tester sous MATLAB les algorithmes classiques d'optimisation étudiés en cours sur une fonction mathématique dédiée.**
- **Mise en application sur un cas pratique à identifier.**

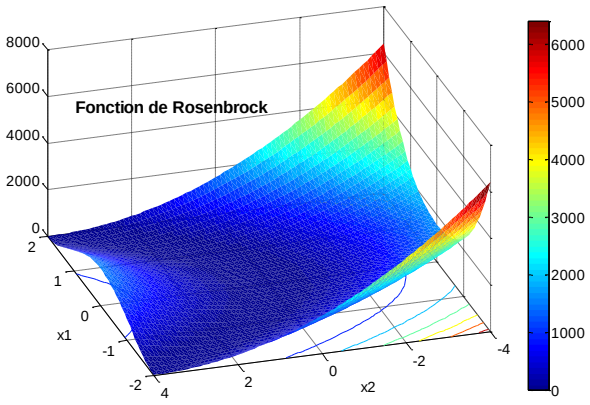
Durée du projet (par groupe de 2 étudiants) : 9h encadrées + Rédaction

Document à rendre sur MOOTSE : rapport écrit 10 pages + Programmes Matlab

Il existe dans la littérature un certain nombre de fonctions mathématiques couramment utilisées pour tester les performances des différentes méthodes d'optimisation. Chacune d'entre elles possède des caractéristiques propres à un type particulier de minimum plus ou moins accessible. Parmi celles-ci on trouve par exemple la fonction ci-dessous proposée par Rosenbrock en 1960 pour montrer la supériorité de son algorithme en comparaison avec la méthode de la plus forte pente :

$$f(x_1, x_2) = 100\left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2$$

Elle possède une vallée qui suit une parabole $x_2 = x_1^2$, ce qui oblige toute méthode de descente à suivre une trajectoire courbe. Le minimum global de cette fonction se situe en $x^* = \begin{pmatrix} 1 & 1 \end{pmatrix}^T$.



1) **Choisissez** une fonction sur https://en.wikipedia.org/wiki/Test_functions_for_optimization parmi celles proposées concernant l'optimisation simple objectif (hors fonction de Rosenbrock). **Représenter** cette fonction dans le domaine de définition cité. Vous pouvez pour cela utiliser les fonctions *surf* ou *mesh*.

Dans ce qui suit, les résultats seront IMPERATIVEMENT affichés en précisant **la valeur du vecteur résultat x_f obtenu** ainsi que **l'évaluation de la fonction coût** en ce point $f(x_f)$. La « qualité » de la solution sera également exprimée par **la mesure d'optimalité du premier ordre** donnée par $\|\nabla f(x_f)\|_\infty = \max_i \left[\left(\nabla f(x_f) \right)_i \right]$

2) **Programmer** la méthode des **plus fortes pentes avec préconditionnement** (algorithme chapitre 3 T15) en utilisant la recherche linéaire pour le pas à chaque itération de façon à s'assurer que les conditions de Wolfe soient bien remplies. La famille de préconditionneurs sera celle correspondante à la matrice identité. Es ce que la méthode converge vers la bonne solution ?

- **Tracer** le cheminement du point courant au cours des itérations sur une courbe de niveau (fonction *contour*).
- **Relancer** avec plusieurs valeurs de paramètres β_1 et β_2. Conclusions
- **Relancer** pour différents points de départ x_0. Conclusions

3) **Programmer** la **méthode de Newton avec recherche linéaire** (algorithme chapitre 3 T23) Tracer comme précédemment le cheminement du point courant à l'itération k sur une courbe de niveau. Conclure quant à l'efficacité de la méthode. Comparer avec la méthode précédente ?

4) Réaliser la même étude avec la **méthode quasi-Newton BFGS.**

- En **programmant** l'algorithme du cours chapitre 3 T29.
- En utilisant la fonction *fminunc* de la **Toolbox Matlab Optimization**
- **Comparer** puis expliquer les différences de performances obtenues avec la méthode de Newton.

5) Les problèmes d'optimisation entourent notre quotidien et nous sommes constamment amenés à les résoudre d'une manière qui n'est pas forcément optimal… **Réfléchir à un problème d'optimisation** que vous avez/allez rencontrer dans votre vie à l'école, en entreprise, dans vos loisirs afin de le mettre en forme mathématiquement suivant la procédure décrite dans le chapitre 1. Vous décrirez de façon très précise les variables de décisions, la fonction objectif ainsi que les contraintes entourant le problème. Une fois cette étape réalisée, **utiliser soit les fonctions de la toolbox Matlab, soit vos propres programmes** pour résoudre le problème. Il est conseillé de choisir un problème sans contraintes (ou très peu). Comparer les différents résultats obtenus (s'il y a lieu).

Aide au déroulement du projet

Certains algorithmes apparaissent dans plusieurs méthodes. Il est recommandé de bien les isoler des programmes afin de pouvoir les réutiliser.

Il est recommandé de tester d'abord les algorithmes sur des exemples simples afin de vérifier leur bon fonctionnement et à défaut les déboguer. Par exemple on peut choisir le problème de minimisation de la fonction $f(x_1, x_2) = \frac{1}{2}x_1^2 + \frac{9}{2}x_2^2$ donné en cours dans chapitre 3 T16.

Il est généralement instructif de faire varier les paramètres intrinsèques des différents algorithmes afin de comprendre leur rôle. Lors de l'implémentation, **il est déconseillé de définir la valeur de ces paramètres dans le code** mais plutôt dans un fichier à part qui rendra la modification plus aisée lors des différents tests.

Il est également commode de faire ressortir des informations relatives à chaque itération (itération courante, valeur de la fonction objectif, norme du gradient, etc..) afin **d'analyser le comportement du programme** et/ou d'identifier une erreur.

Il faut être vigilent en ce qui concerne les **problèmes numériques**. En effet, les ordinateurs fonctionnent en arithmétique dite finie, en ce sens que seulement un nombre fini de réels peut être représenté. Une des conséquences est la prise en compte de la plus petite valeur possible appelée le ε-machine (variable *eps* sous MATLAB) telle que $1+\varepsilon \neq 1$.

La **prise en compte des erreurs** est très importante. Si l'algorithme tente d'inverser une matrice singulière, il faut pouvoir le détecter « proprement » et afficher un message d'erreur adéquat.

Bien que les conditions d'arrêt des algorithmes décrits en cours soient réalisées par un test sur la condition d'optimalité du 1er ordre, il sera plus prudent de **fixer également un nombre maximal d'itérations** afin de forcer le programme à s'arrêter en cas d'erreur et/ou de défaut de convergence.

Afin d'étudier les performances des divers algorithmes, il faut choisir une (ou plusieurs mesures) qui soit comparable d'un algorithme à l'autre. Il s'agit généralement du **temps de calcul**, du **nombre d'itérations**, du **nombre d'évaluations** de la fonction objectif. Par exemple, si 2 algorithmes convergent vers des solutions différentes, cela n'a pas de sens de comparer le nombre d'itérations !